Trees

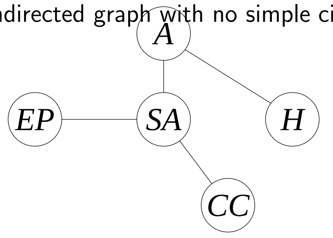
Algorhyme by Radia Perlman

I think that I shall never see
A graph more lovely than a tree.
A tree whose crucial property
Is loop-free connectivity.
A tree which must be sure to span.
So packets can reach every LAN.
First the Root must be selected
By ID it is elected.
Least cost paths from Root are traced
In the tree these paths are placed.
A mesh is made by folks like me
Then bridges find a spanning tree.

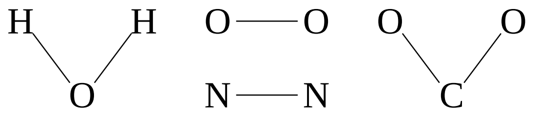
Definitions

Trees

A *tree* is a connected undirected graph with no simple circuits.

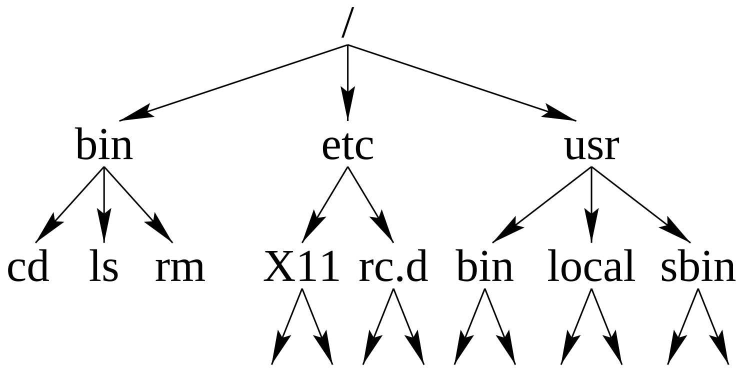


A *forest* is a undirected graph with no simple circuits. A forest is a set of trees.



Rooted Trees 1

In a *rooted tree*, one vertex is the *root*, and all edges are directed away from the root.



Rooted Trees 2

If there is an edge (u, v) in a rooted tree,
u is the *parent* of v, and v is a *child* of u.

If u has children, then u is *internal*.
If u has no children, then v is a *leaf*.
u and v are *siblings* if they have same parent.

The *height* of a tree is the length of the longest path from the root to a vertex.

The *level* of a vertex is the path length from the root. The root is on *level* 0.

Rooted Trees 3

If there is a path from u to v in a rooted tree,
u is an *ancestor* of v, v is a *descendent* of v, and v is in u's *subtree*.

A rooted tree is an m*-ary* tree if every internal vertex has $\leq m$ children. It is a *full* m-ary tree if every internal vertex has exactly m children.

A *binary* tree is an m-ary tree with $m = 2$.

In an *ordered rooted tree*, siblings are ordered. In an ordered binary tree, an internal vertex has a *left* child and/or a *right* child.

Properties of Trees 1

A tree with n vertices has $n - 1$ edges.
Proof: Every vertex except the root has an edge from its parent.

An m-ary tree with i internal vertices has at most $mi + 1$ vertices.
Proof: Each internal vertex can at most m edges to its children. mi edges imply $mi + 1$ vertices.

A binary tree of height h has at most 2^h leaves. Proof later.

Properties of Trees 2

A binary tree of height h has at most $2^{h+1} - 1$ vertices. Adapt proof of previous property.

The minimum height of a binary tree with l leaves is $h = \lceil \log_2 l \rceil$.
Proof:
Minimum height h must satisfy $2^{h-1} < l \leq 2^h$.
This is equivalent to $h - 1 < \log_2 l \leq h$.
It follows that $\lceil \log_2 l \rceil = h$

The minimum height of a binary tree with n vertices is $\lfloor \log_2 n \rfloor$. Adapt previous proof.

Number of Leaves in a Binary Tree

A binary tree of height h has at most 2^h leaves.

Basis: A binary tree of height 0 has $2^0 = 1$ leaf.

Induction: Assume $k \geq 0$ and a binary tree of height k has at most 2^k leaves.
Show a binary tree of height $k+1$ has at most 2^{k+1} leaves.

Proof: The root of a binary tree of height $k+1$ can have 2 subtrees of height k (or less).
By the inductive assumption, a binary tree of height k has at most 2^k leaves.
2 subtrees of height k have at most $2(2^k) = 2^{k+1}$ leaves.

Binary Search Trees

In a *binary search tree*, each vertex is labeled with a key, the left tree of a vertex contains smaller keys, and its right tree contains larger keys.

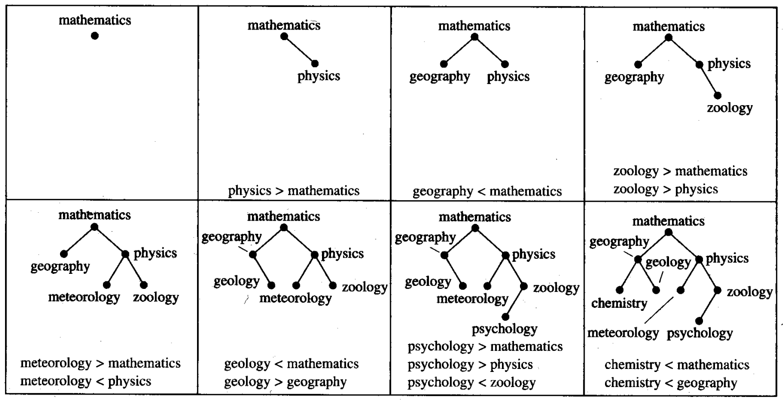


FIGURE 1 Constructing a Binary Search Tree.

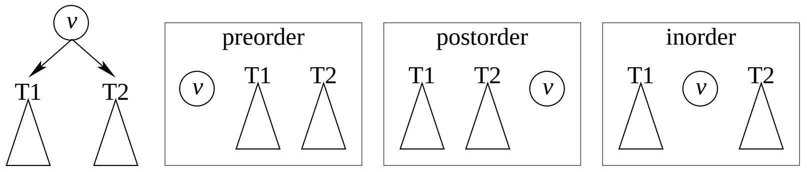
Tree Traversal 1

```
procedure $preorder(v)$
    assert $v$ is a vertex in an ordered rooted tree
    list $v$
    for each child $c$ of $v$ from left to right
        $preorder(c)$

procedure $postorder(v)$
    assert $v$ is a vertex in an ordered rooted tree
    for each child $c$ of $v$ from left to right
        $postorder(c)$
    list $v$
```

Tree Traversal 2

```
procedure $inorder(v)$
    assert $v$ is a vertex in an ordered rooted tree
    for each child $c$ of $v$ from left to right
        $inorder(c)$
        if $c$ is first child of $v$ then list $v$
    if $v$ has no children then list $v$
```



Expression Trees

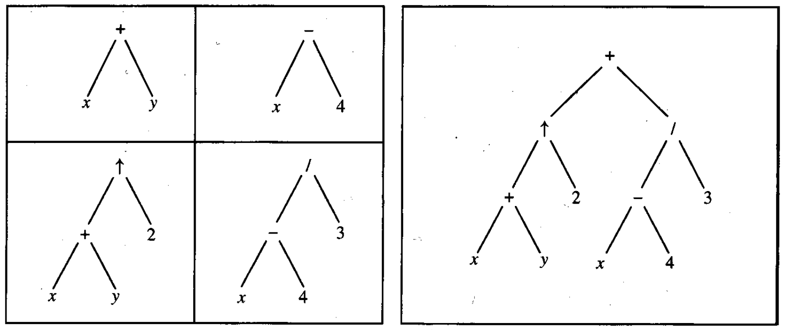


FIGURE 10 A Binary Tree Representing $((x + y) \uparrow 2) + ((x - 4)/3)$.

More Expression Trees

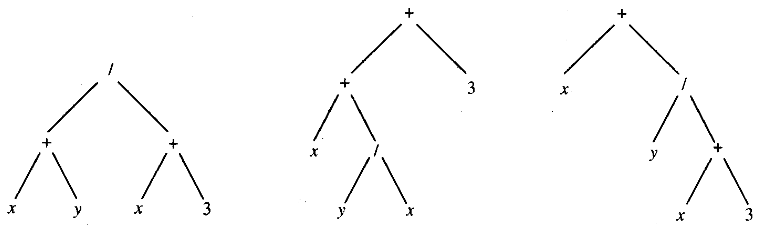


FIGURE 11 Rooted Trees Representing $(x + y)/(x + 3)$, $(x + (y/x)) + 3$, and $x + (y/(x + 3))$.

Spanning Trees

Spanning Trees

A *spanning tree* of a simple graph G is a subgraph that is a tree and contains every vertex.

A simple graph is connected if and only if it has a spanning tree.

Depth-first search and breadth-first search can be used to find spanning trees.

Depth-first and breadth-first search can be implemented in $O(v + e)$ time where v is the number of vertices and e is the number of edges.

Depth-First Search Again

```
procedure $DFS$ $(G)$
   assert $G$ is weighted connected undirected graph
   $stack$ := arbitrary vertex in $G$
   $T$ := empty tree
   while $stack$ is not empty
     assert $x \in T$ or $x$ reachable from some $y \in stack$
     $u$ := top of $stack$
     if there is an edge $\{u, v\}$ s.t. $v \notin T$ then
       add $\{u, v\}$ to $T$ and push $v$ on $stack$
     else pop $stack$
   end while
   assert all vertices are in $T$
   return $T$
```

Breadth-First Search

```
procedure $BFS$ $(G)$
   assert $G$ is weighted connected undirected graph
   $queue$ := arbitrary vertex in $G$
   $T$ := empty tree
   while $queue$ is not empty
     assert $x \in T$ or $x$ reachable from some $y \in queue$
     $u$ := beginning of $queue$
     if there is an edge $\{u, v\}$ s.t. $v \notin T$ then
       add $\{u, v\}$ to $T$ and enqueue $v$ on $queue$
     else dequeue $queue$
   end while
   assert all vertices are in $T$
   return $T$
```

Minimum Spanning Trees

A *minimum spanning tree* of a weighted simple graph G is a spanning tree with the minimum sum of weights.

Prim's algorithm and Kruskal's algorithm can be used to find minimal spanning trees.

Prim's algorithm and Kruskal's algorithm can be implemented in $O(e \log e)$ time where e is the number of edges.

Prim: Find min. edge from tree so far to new vertex.
Kruskal: Find minimum edge avoiding cycle.

Prim's Algorithm

```
procedure $Prim$ $(G)$
    assert $G$ is weighted connected undirected graph
    $T$ := smallest edge in $G$ and the edge's vertices
    $n$ := number of vertices in $G$
    for $i$ := 1 to $n - 2$
        assert $T$ is a tree with $i$ edges
        $e$ := smallest edge in $G$ with one vertex
               in $T$ and the other vertex not in $T$
        add $e$ and the new vertex to $T$
    end for
    assert $T$ is a tree with $n - 1$ edges
    return $T$
```

Kruskal's Algorithm

```
procedure $Kruskal$
                                        ($G$: weighted connected simple graph)
    assert $G$ is weighted connected undirected graph
    $T :=$ vertices of $G$ and smallest edge in $G$
    $n :=$ number of vertices in $G$
    for $i := 1$ to $n - 2$
        assert $T$ is a forest with $n - i$ components
        $e :=$ smallest edge in $G$ connecting
               two components of $T$
        add $e$ to $T$
    end for
    assert $T$ is a forest with $1$ component
    return $T$
```